

Bash Scripts





The Basic Steps

1. Write a script in a file and save it
2. Make the script executable using `chmod`
3. Verify that the shell can find your script





Shebang!

The first line of our script should read `#!/bin/bash`

The `#!` is called the shebang, and it's used to tell the OS which interpreter it should use when parsing this file.

We want ours to say "use bash to interpret this file!"

After the shebang, we need to include the path to the Bash binary. This is not Bash specific. If we wanted to write a python script, we would include the path to the python binary.

```
#!/bin/bash
```

```
#my first script
```

```
echo "hello everyone"
```





Comments

Lines that begin with `#` will not be read by the shell.
Write comments to explain particularly tricky bits of code or to leave notes for yourself.

```
#!/bin/bash
```

```
#my first script
```

```
echo "hello everyone"
```





The Good Stuff

We can write any of the commands that we normally run from the command line. When we execute the script, these commands will run!

```
#!/bin/bash
```

```
#my first script
```

```
echo "hello everyone"
```





Executing The Script

We can execute the script the "long way" by running **bash pathToFile**.

This works, but it's not as convenient as it could be! What if we could instead just run **hello** from anywhere on our machine, just like we can run **ls** or **grep** anywhere?



```
> bash ~/scripts/hello
```





Locating Commands

When we run a command like `pwd`, the shell starts looking for the executable `pwd` program in the list of directories stored in the **PATH** variable.

It starts looking in the first location and then keeps looking if it doesn't find it.

A dark blue terminal window with three colored window control buttons (red, yellow, green) in the top left corner. A light blue prompt character is followed by the command `pwd`.

```
>pwd
```

A magnifying glass with a yellow circle and a pink handle. The handle is positioned to the left of the first directory in the list, pointing towards it.

```
/usr/local/sbin
```

```
/usr/local/bin
```

```
usr/bin
```





Locating Commands

When we run a command like `pwd`, the shell starts looking for the executable `pwd` program in the list of directories stored in the `PATH` variable.

It starts looking in the first location and then keeps looking if it doesn't find it.

A dark blue rounded rectangle representing a terminal window, with three colored dots (red, yellow, green) in the top-left corner.

```
>pwd
```

~~/usr/local/sbin~~

A magnifying glass with a yellow circle and a pink handle, positioned over the text.

/usr/local/bin

usr/bin





Locating Commands

When we run a command like `pwd`, the shell starts looking for the executable `pwd` program in the list of directories stored in the `PATH` variable.

It starts looking in the first location and then keeps looking if it doesn't find it.



```
>pwd
```

~~/usr/local/sbin~~

✓ /usr/local/bin

usr/bin





Why It Matters

If we want the shell to find our own programs, we need to make sure we put them in a folder that is in the **PATH** variable.

A common place to put user-defined programs is in a bin folder located in the user's home directory. For me that would be **/home/colt/bin**.

If that directory is not yet part of your path, you can add it by putting **PATH="\$HOME/bin:\$PATH"** in your .bashrc file



```
PATH="$HOME/bin:$PATH"
```

If ~/bin is not yet in your PATH, add the above line to your .bashrc file





Making It Executable

The next step is to make the file containing our script executable. `chmod a+x file` will grant executable permissions to everyone.

A dark blue terminal window with three colored dots (red, yellow, green) in the top left corner. Inside the window, a light blue prompt character is followed by the command `chmod a+x file` in white text.

```
>chmod a+x file
```

